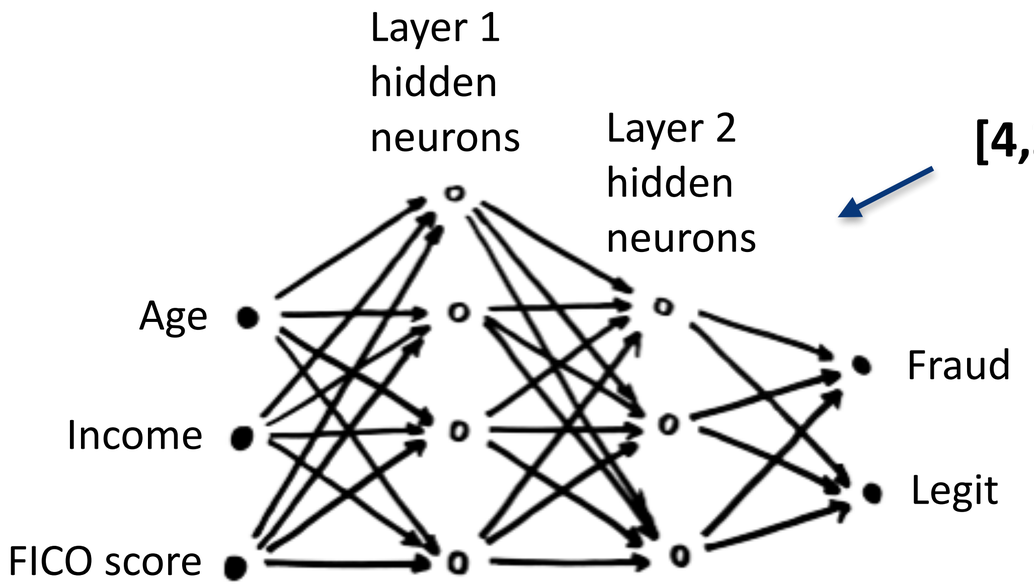
# Top ~~10~~ 13 Deep Learning Tips & Tricks

Arno Candel, PhD
Chief Architect, H2O.ai
@ArnoCandel

H2O WORLD 2015

# Tip #1: Understand Model Complexity




- Neurons are connected layer-by-layer, feed-forward, directed acyclic graph
- Each connection is a floating-point number (weight)
- Neuron activation/output is non-linear transform of weighted sum of inputs
- Model complexity (training time) grows with total number of weights
- Training is iterative: can stop at any point and continue until convergence
- Multi-threading leads to race conditions, results are not reproducible

CPU intense

# Tip #1: Understand Model Complexity

**Given data:** 10 billion rows, 1111 columns
**1110 predictor columns (1100 numerical, 10 categorical with 400 levels each), 1 binary response column**

**Question:** Which model is more complex (i.e., is bigger, has more weights)?

Model 1: **[400,400,400,400]** hidden neurons

Model 2: **[500]** hidden neurons

# Tip #1: Understand Model Complexity

**Given data:** 10 billion rows, 1111 columns
**1110 predictor columns (1100 numerical, 10 categorical with 400 levels each), 1 binary response column**

**Answer**: Model 2. Why?
Categoricals are one-hot expanded into 4000 “dummy” variables (0/1), and hence there are a total of 1100 + 4000 = **5100 input neurons.**

Model 1: **[5100,400,400,400,400,2]** neurons and
**5100x400** + 400x400 + 400x400 + 400x400 + 400x2 = **2.52M** weights

Model 2: **[5100,500,2]** neurons and **5100x500** + 500x2 = **2.55M** weights

Model size depends on features

# Tip #1: Understand Model Complexity

**Given data:** 10 billion rows, 1111 columns
**1110 predictor columns (1100 numerical, 10 categorical with 400 levels each), 1 binary response column**

**Question:** How much memory does H2O Deep Learning need to train such a model (2.5M weights) on 10 billion rows for 100 epochs?

**Answer:** **Model memory usage is small and constant over time:**
2.5M x 4Bytes (float) x 3 (adaptive learning rate) = **30MB**
(more training -> hopefully better weights, but not more weights)

Model size independent of #rows or training time

# Tip #2: Establish a Baseline on Holdout Data

For a given dataset (**with proper holdout sets**), do the following first:

- **Train default GLM, DRF, GBM models** and inspect convergence plots, train/test metrics and variable importances
- **Train default DL models**, but set hidden neurons (and maybe epochs):
  - **hidden=[200,200]** "I Feel Lucky"
  - **hidden=[512]** "Eagle Eye"
  - **hidden=[64,64,64]** "Puppy Brain"
  - **hidden=[32,32,32,32,32]** "Junior Chess Master"
- **Develop a feel for the problem and the holdout performance of the different models**

Develop a Feel

# Tip #3: Inspect Models in Flow

- Inspect the model in **Flow** during training: **getModel 'model_id'**
- If the model is **wrong** (wrong architecture, response, parameters, etc.), cancel it
- If the model is **taking too long**, cancel and decrease model complexity
- If the model is **performing badly**, cancel and increase model complexity

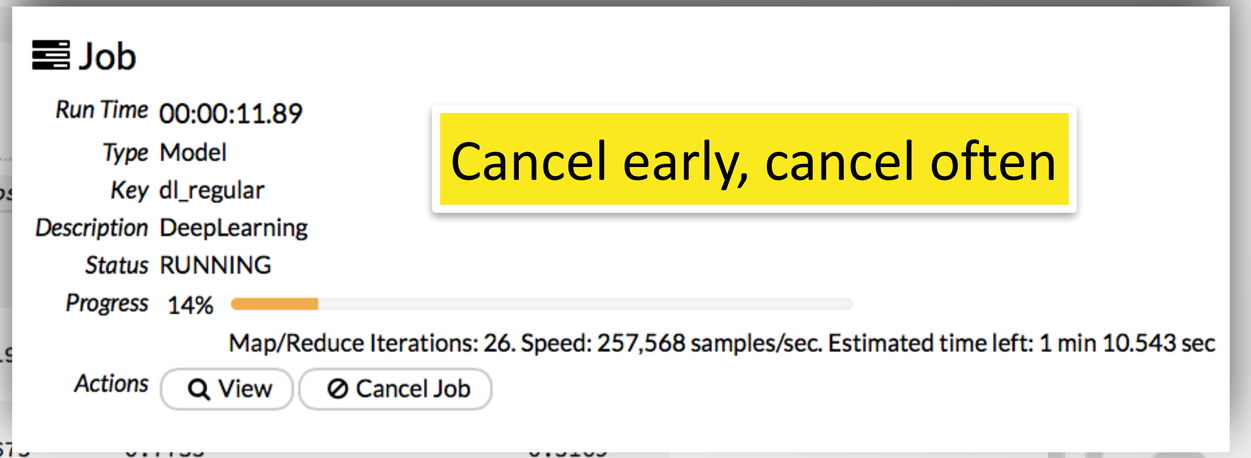
▸ OUTPUT

▾ OUTPUT - STATUS OF NEURON LAYERS (PREDICTING C1, 2-CLASS CLASSIFICATION, BERNOULLI DISTRIBUTION, CROSSENTROPY LOSS, SIZE 1)

| layer | units | type | dropout | l1 | l2 | mean_rate | rate_RMS | momentum | mean_weight | weight_RMS | mean_bias | bias_RMS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 28 | Input | 0 | | | | | | | | | |
| 2 | 50 | RectifierDropout | 30.0 | 0.0001 | 0.0001 | 0.0016 | 0.0009 | 0 | -0.0120 | 0.2272 | -0.1812 | 0.2313 |
| 3 | 50 | RectifierDropout | 30.0 | 0.0001 | 0.0001 | 0.0075 | 0.0165 | 0 | -0.1070 | | | |
| 4 | 2 | Softmax | | 0.0001 | 0.0001 | 0.0048 | 0.0030 | 0 | 0.0876 | | | |

▾ OUTPUT - SCORING HISTORY

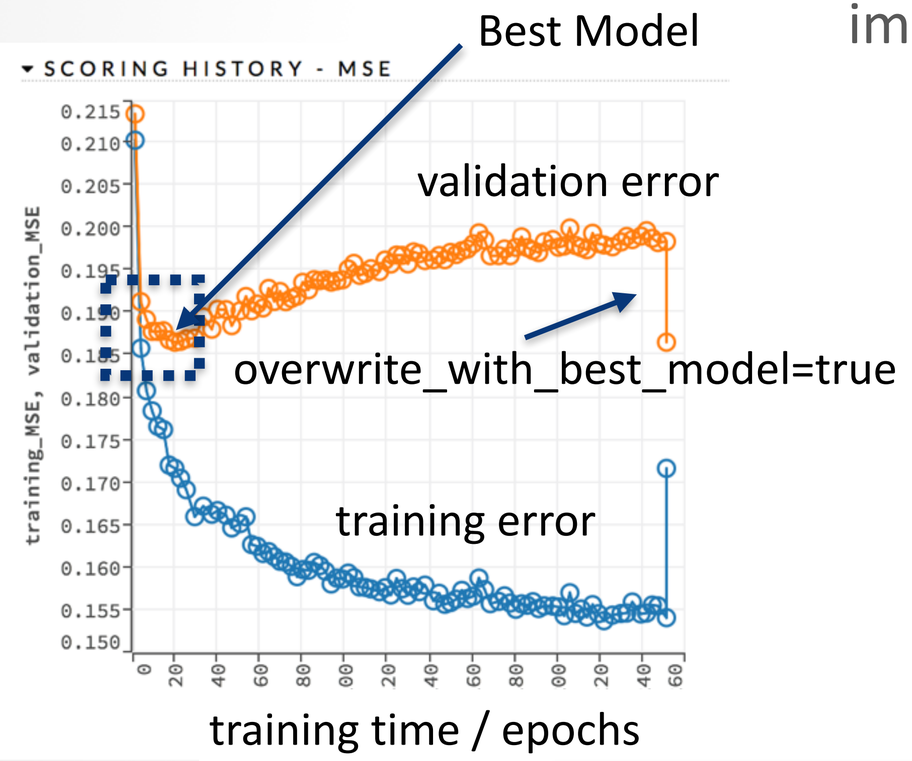
| timestamp | duration | training_speed | epochs | samples | training_MSE | training_r2 | training_loglos | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2015-11-08 23:16:58 | 0.000 sec | | 0 | 0 | . | . | | | | | |
| 2015-11-08 23:16:59 | 0.758 sec | 161733 rows/sec | 1.3330 | 99951.0 | 0.2155 | 0.1352 | 0.619 | | | | |
| 2015-11-08 23:17:01 | 2.441 sec | 183203 rows/sec | 5.3313 | 399749.0 | 0.1935 | 0.2235 | 0.567 | | | | |
| 2015-11-08 23:17:02 | 3.827 sec | 200821 rows/sec | 9.3392 | 700263.0 | 0.1885 | 0.2437 | 0.5554 | 0.7881 | | | 0.3155 |



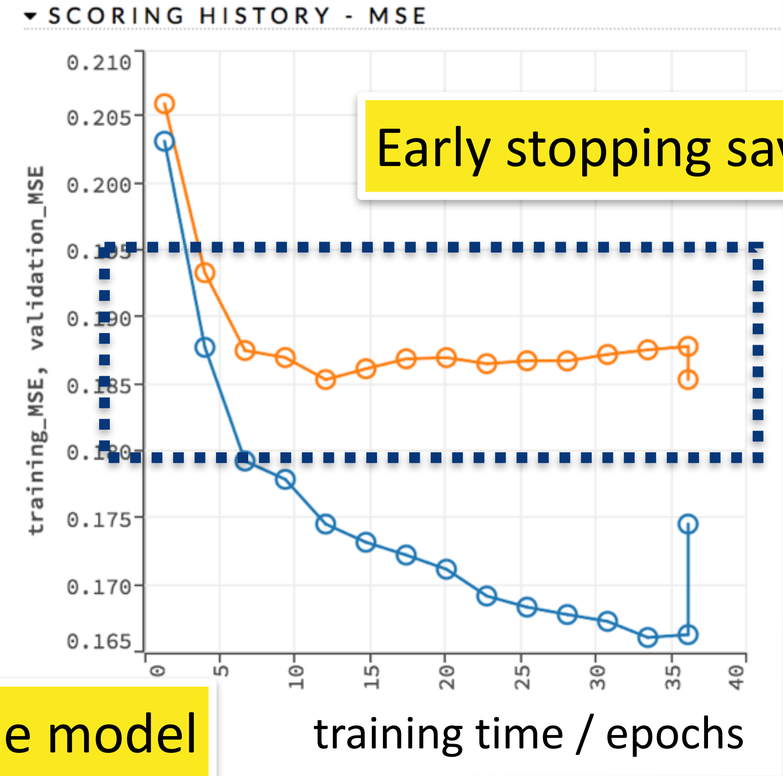

H2O WORLD

# Tip #4: Use Early Stopping! (On by default)

**Before:** trains too long, but at least **overwrite_with_best_model=true** prevents overfitting (returns the model with lowest validation error)




**Now:** specify additional convergence criterion: E.g. **stopping_rounds=5, stopping_metric="MSE", stopping_tolerance=1e-3,** to stop as soon as the moving average (length 5) of the validation MSE does not improve by at least 0.1% for 5 consecutive scoring events




Use Flow to inspect the model

Higgs dataset

# Tip #5: Control Scoring Overhead

By default, scoring is limited to 10% of the runtime, but if you provide a large validation dataset, then even scoring for the first time can take a long time. Here are options to control this overhead:

- **Provide a smaller validation dataset** for scoring during training
- **Sample the validation dataset** with 'score_validation_samples=N'
  - If multi-class or imbalanced, set 'score_validation_sampling="Stratified"'
- **Score less often** with 'score_duty_cycle=0.05' etc.

The quality of the validation dataset and the frequency of scoring can affect early stopping. Especially if N-fold cross-validation is disabled, try to provide a validation dataset for accurate early stopping.

Validation data determines scoring speed and early stopping decision

# Tip #6: Use N-fold Cross-Validation

In H2O, N-fold Cross-Validation trains N+1 models:

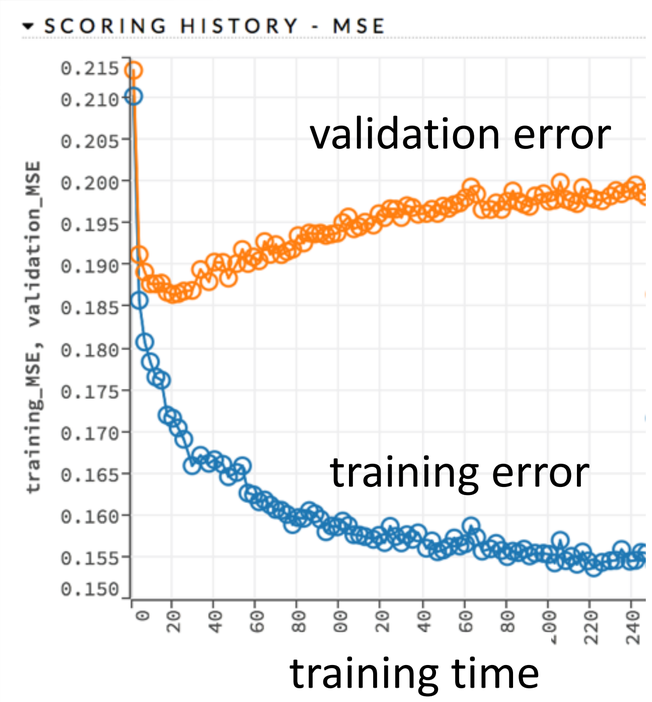
- N models with each a different 1/N-th of the training data held out as validation data
- 1 model on the full training data

Early stopping and N-fold CV are especially useful together, as the optimal number of epochs is estimated from the N Cross-Validation models (even if a validation dataset is provided).

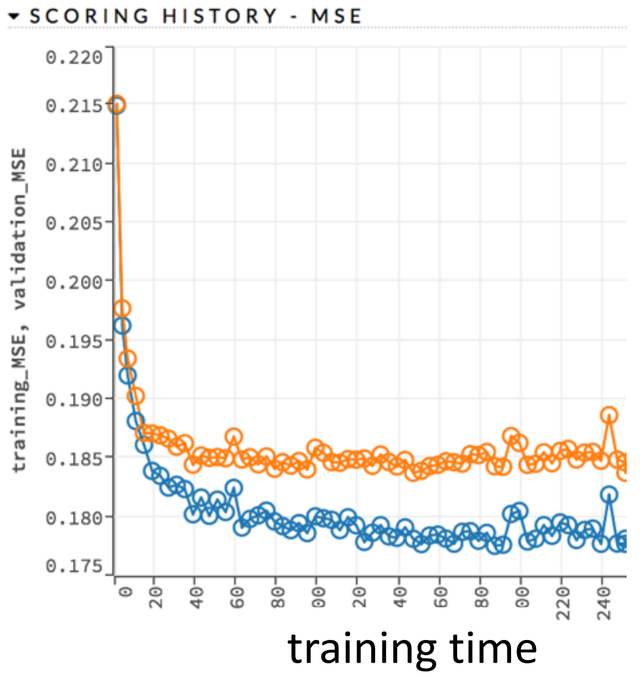
- **Early stopping automatically tunes the number of epochs**
- **Model performance is estimated with training holdout data**
- **The model is built on all of the training data**

Estimate your model performance well

# Tip #7: Use Regularization




**Rectifier**, hidden=[50,50]




**RectifierWithDropout**, hidden=[50,50],
**l1=1e-4, l2=1e-4, hidden_dropout_ratio=[0.2,0.3]**

Overfitting is easy, generalization is art

Higgs dataset

# Tip #8: Perform HyperParameter Search

There are a lot of hyper parameters for H2O Deep Learning. Many are for expert-level fine control and do not significantly affect the model performance. **Rectifier** / **RectifierWithDropout** is recommended for most problems (The only difference is the default values for hidden_dropout_ratios: 0 / 0.5).

Main parameters to tune (I prefer **Random** over **Grid search**) are:

- **hidden (try 2 to 5 layers deep, 10 to 2000 neurons per layer)**
- **hidden_dropout_ratios** and **input_dropout_ratio**
- **l1/l2**
- **adaptive_rate**
  - **true**: **rho, epsilon**
  - **false**: **rate, rate_annealing, rate_decay, momentum_start, momentum_stable, momentum_ramp**

Just need to find one of the many good models

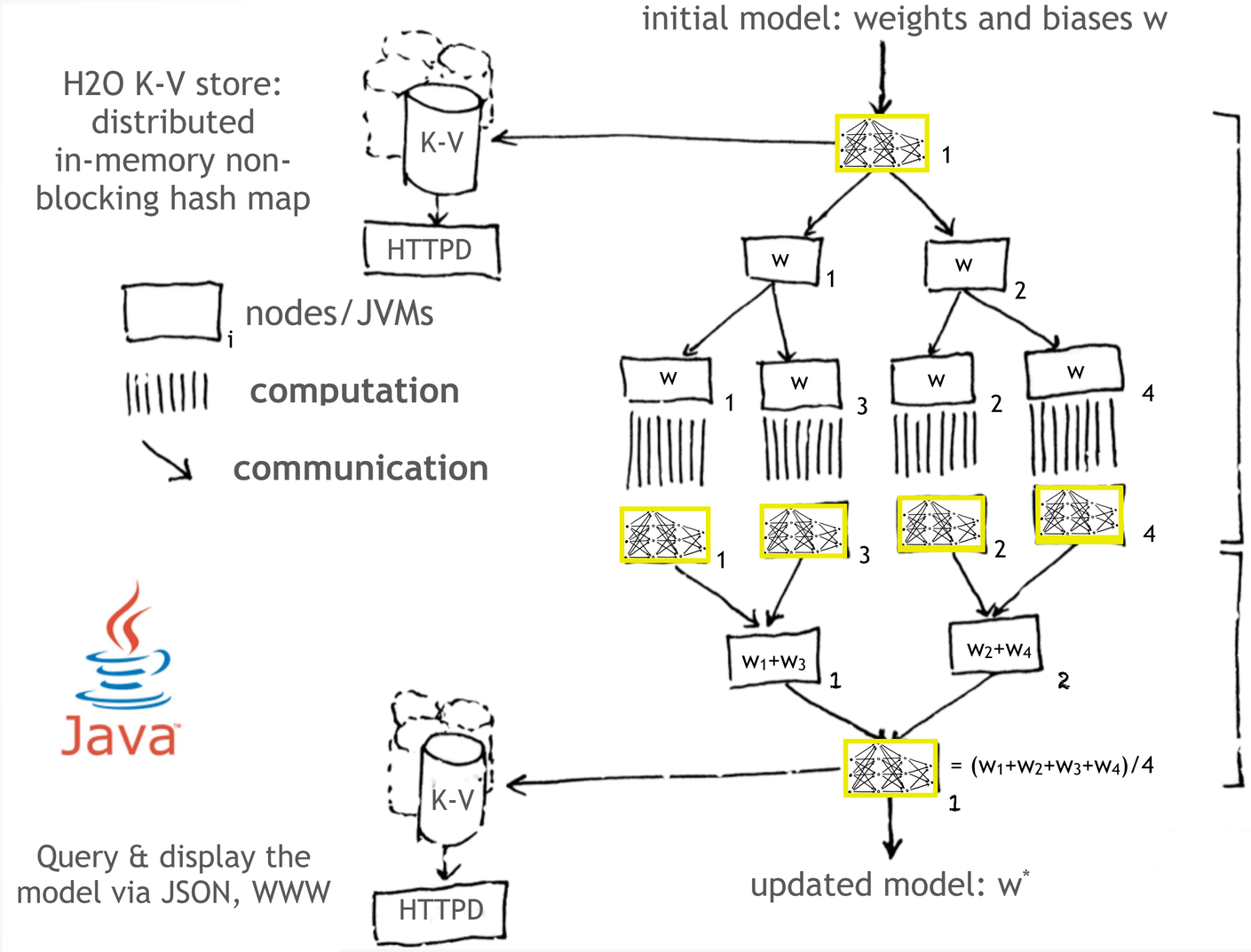
# Tip #9: Use Checkpointing

Any H2O Deep Learning model can be stored to disk and then later restarted, with either the same or new training data.

- It's **ok to cancel** a H2O Deep Learning model at any time during training, it can be restarted
- Store your favorite models to **disk** for later use
- **Continue training from previous model checkpoints** (architecture, data types, etc. must match, many parameters are allowed to change)
- Start multiple new models from the same checkpoint, with different regularization, learning rate parameters to **speed up hyper-parameter search**

Checkpointing enables fast exploration

# Tip #10: Tune Communication on Multi-Node




## Multi-Node

**Map: training**
each node trains independently, parallelizes well

**Reduce: model averaging**
can improve accuracy, but takes time (limited to ~5% of time by default)

Know your ‘target_ratio_comm_to_comp’ value

# Tip #11: Know your Data (Sparsity/Categoricals)

If the number of input neurons is >> 1000, then training can be inefficient and slow, especially if there are many sparse features (e.g., categorical predictors) that rarely activate neurons. **It might be more efficient to train faster with fewer (dense, rich) predictors.**

Try the following remedies:

- Build a tiny DL model and use **h2o.deepfeatures**() to extract lower-dimensional features
- **Ignore categorical columns** with high factor counts
- Use **random projection** of all categorical features into N-dimensional space, by setting **'max_categorical_features=N'**
- Use **GLRM to reduce the dimensionality of the dataset** (and make sure to apply the same GLRM model to the test set)
- If the dataset is just simply wide (few or no categorical features), then the chance of it being sparse is high. If so, set '**sparse=true**', which can result in faster training.

Know your data

# Tip #12: Advanced Math

For regression problems, it might make sense to apply different loss functions and distributions:

- **Gaussian** distribution, squared error loss, sensitive to outliers
- **Laplace** distribution, absolute error loss, more robust to outliers
- **Huber** loss, combination of squared error & absolute error
- **Poisson** distribution (e.g., number of claims in a time period)
- **Gamma** distribution (e.g, size of insurance claims)
- **Tweedie** distribution (compound Poisson-Gamma)

Also, H2O Deep Learning supports:

- **Offsets (per-row)**
- **Observation weights (per-row)**

Know your math

H2O WORLD

# Tip #13: Do Ensembling

To obtain highest accuracy, it's often more effective to average a few fast and diverse models than to build one large model.

Ideally, use different network architectures, regularization schemes to keep the variance high before ensembling.

Given a set of (diverse, and ideally, good or great) models, use

- **Blending** (just average the models)
- Gut-feel Blending (assign your own weights that add up to 1)
- **SuperLearner** (optimized **stacking** with meta-learner, see Erin's talk)
- **Add other models** into the mix (GBM, DRF, GLM, etc.)

Ensembles rule the leaderboards

# Summary: Checklist for Success with H2O Deep Learning

- Understand Model Complexity
- Inspect Models in Flow
- Control Scoring Overhead
- Use Regularization
- Use Checkpointing
- Know your Data (Sparsity/Categoricals)
- Establish Baseline on Holdout Data
- Use Early Stopping
- Use N-fold Cross-Validation
- Perform HyperParameter Search
- Tune Communication on Multi-Node
- Know your Math
- Do Ensembling